스마트 폰을 활용한 Smart Air IoT-Cloud 응용 서비스

Document No. 1
Version 1.0
Date 2017-11-30
Author(s) 김청빈, 김준혁

■ **문서의 연혁**

버전	날짜	작성자	비고
초안 - 0.1	2017. 09. 20	김청빈	
중간본 - 0.2	2017. 11. 9	김준혁, 김청빈	
중간본 - 0.3	2017. 11. 25	김청빈, 김준혁	
최종본 - 1.0	2017. 11. 30	김청빈, 김준혁	

본 연구는 한국정보화진흥원(NIA)의 미래네트워크선도시험망 (KOREN) 사업 지원과제의 연구결과로 수행되었음 (17-951-00-001).
This research was one of KOREN projects supported by National Information Society Agency (17-951-00-001).

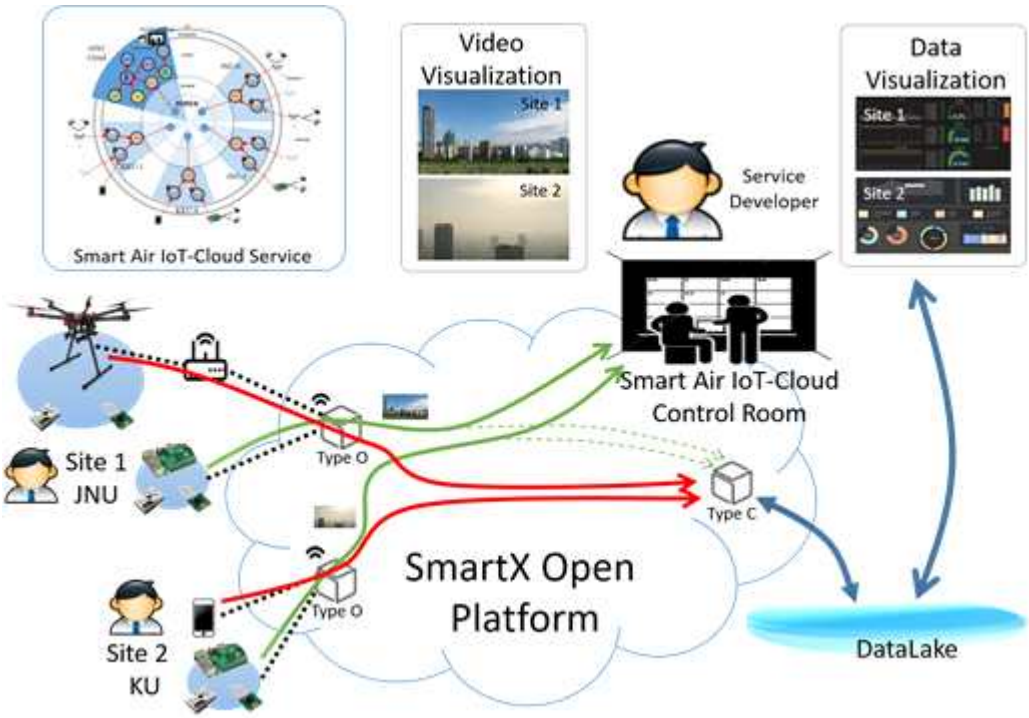
Contents

#8. 스마트 폰을 활용한 Smart Air IoT-Cloud 응용 서비스

그림 목차

#8. 스마트 폰을 활용한 Smart Air IoT-Cloud 응용 서비스

1. 개요



<그림 1: Smart Air IoT-Cloud 응용 서비스>

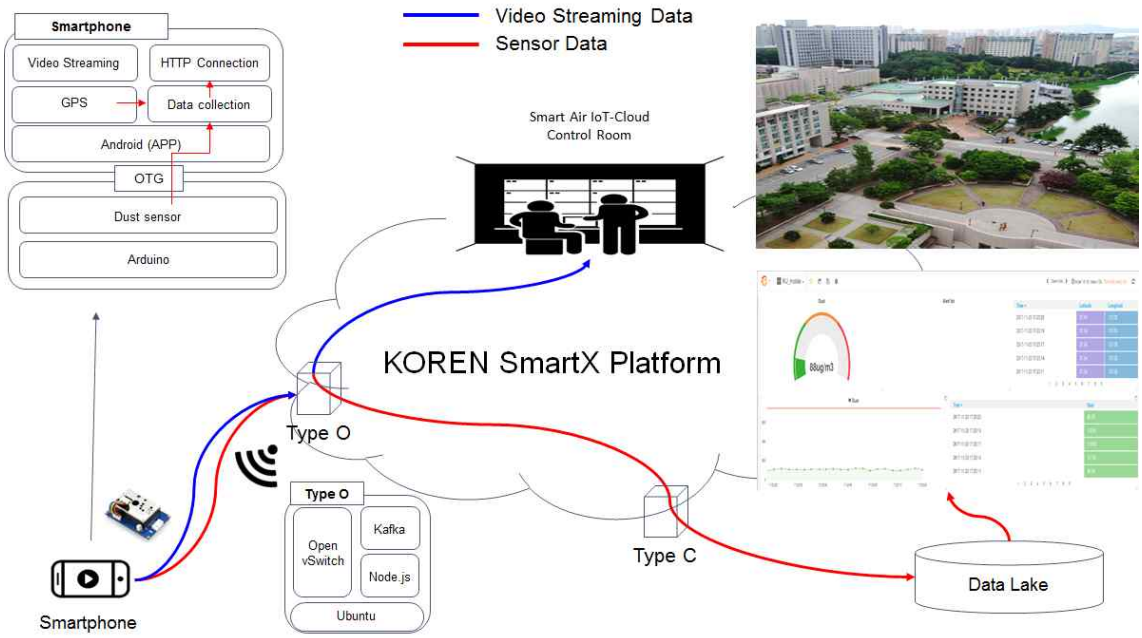
o 본 문서에서는 KOREN SmartX Open Platform에서 제공하는 SDN/SDI OpenAPI를 활용하여 사용자가 원하는 서비스 운영에 필요한 클라우드/네트워크 자원을 확보하고 추가적인 요소기능을 투입하여 전체 서비스로 합성하는 일련의 응용서비스 지원 프로세스 실증에 대한 내용을 다룬다.

o Smart Air IoT-Cloud 서비스는 다수지역의 기상정보 및 미세먼지 정도를 고정식 센서 모듈 또는 드론 및 스마트 폰을 활용한 이동식 센서 모듈을 통해 실시간으로 측정하여 관련 데이터를 수집하고 관측내용을 분석하여 시각화하는 서비스이다.

o 다수의 사이트에서 운용되는 및 센서 모듈들에서 측정되는 실시간 센싱 데이터는 Smart Air IoT-Cloud서비스를 위한 네트워크 슬라이스에 포함된 Type O SD-Access Box와 연결하여 DataLake로 전송한다. DataLake는 다수의 IoT기기에서 제공되는 데이터를 검증하고 수집하기 위한 Smart IoT-Cloud Hub와 수집된 데이터의 저장 및 간략한 분석기능을 제공하는 SmartX Cloud로 구성된다. DataLake에 수집된 데이터를 Smart Air IoT-Cloud Control Room에서 효과적으로 분석하고 활용하기 위해, 현재 운용중인 서비스 모듈에서의 데이터 흐름 및

상태현황을 손쉽게 파악하는 기능을 제공하는 서비스운용 시각화 기능 및 IoT데이터 통계 분석을 다양한 측면에서 지원하는 데이터분석 UI도구를 제공한다.

2. 데모 시나리오



<그림 2: 데모 구성도>

2.1. 먼지 센서

o 스마트 폰은 어플리케이션을 통해 외부에 장착된 먼지 센서로부터 dust 값을 읽고 내부의 GPS 센서로 부터 Latitude, Longitude 값을 읽는다.

o 읽어온 값들은 JSON 형식으로 파싱하여 HTTP 통신을 사용하여 Node.js로 구현한 서버로 전송한다.

o 서버는 받아온 값을 kafka producer를 통해 Data Lake로 전송한다.

2.2. 스트리밍

o GoCoder SDK를 사용해 개발한 어플리케이션을 통해 촬영한 영상 데이터를 RTMP를 활용해 Wowza Streaming Engine Server로 보내고 Control Room의 Demo visualization에서 비디오 스트림을 호출한다.

3. Smart Air IoT-Cloud 응용 서비스를 센서 데이터 수집

3.1. 개요

o Smart Air IoT-Cloud 서비스 실증 데모를 위해 이동형 IoT기기(스마트 폰) 전용 애플리케이션을 개발한다. 안드로이드용 애플리케이션으로 작성하였고, 실시간 비디오 전송과 센서 데이터 수집 및 전송 기능을 한다. 실시간 비디오 전송은 촬영 중인 영상을 KOREN SmartX Platform 상의 Control Room으로 전송하여 볼 수 있도록 한다. 센서 데이터는 먼지 센서 데이터, GPS 센서 데이터(위, 경도 값)이 있다. 먼지 센서 데이터는 미니 보드와 먼지센서를 통해 제작하여 스마트 폰에 직접 연결 후 수집한다. 위치 정보의 경우 스마트 폰에 내장 된 센서 데이터 값을 수집한다. 그리고 수집한 데이터를 JSON형식으로 파싱하고 HTTP Connection을 이용해 센서 데이터 값들을 Node.js로 구현된 서버로 전송하고 전송된 데이터들은 서버 내부에 구현된 Kafka High Level Producer를 통해 Kafka 브로커로 전송한다.

3.2. 스마트 폰 전용 먼지센서 제작

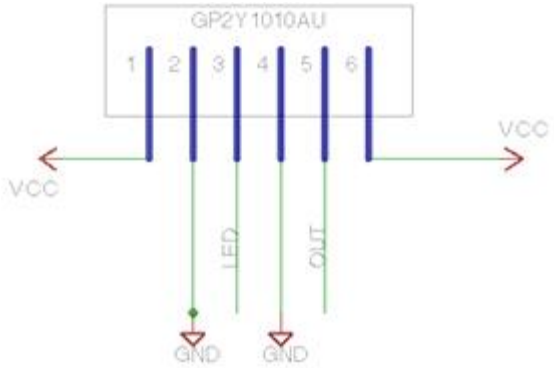
o 스마트 폰을 통한 먼지센서 데이터 수집을 위해 스마트 폰 전용 먼지센서를 제작한다. 스마트 폰에 직접 부착하기 위해서 크기가 작은 모듈들이 필요하다. 보드는 ATTiny85를 사용하고 먼지센서로 GP2Y1010AU를 사용한다.

<그림 3: ATtiny85>

<그림 4: GP2Y101AU>

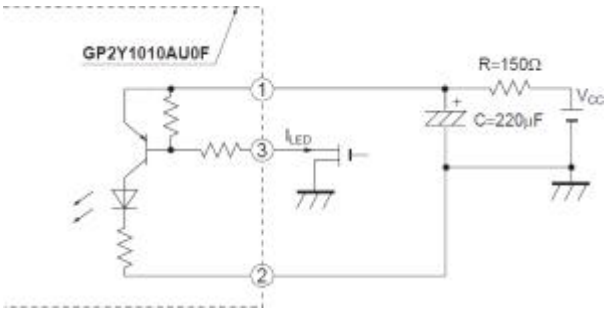
o 스마트 폰을 통한 먼지센서 데이터 수집을 위해 스마트 폰 전용 먼지센서를 제작한다. 스마트 폰에 직접 부착하기 위해서 크기가 작은 모듈들이 필요하다. 보드는 ATTiny85를 사용하고 먼지센서로 GP2Y1010AU를 사용한다.

o GP2Y1010AU 센서는 6개의 핀을 가지고 있다. 1~3번 pin은 적외선 LED 컨트롤 핀이고 나머지 3개의 핀은 Signal 출력 핀이다.



<그림 5: GP2Y1010AU 핀 설명>

o 1~3번 pin을 연결 할 때에는 <그림 6>과 같이 150ohm 저항과 220uF 커패시터를 연결한다.



<그림 6: GP2Y1010AU 회로도>

o ATTiny85와 연결은 다음과 같다.

GP2Y1010AU	ATTiny85
1 V-Led	5V(150ohm저항)
2 LED-GND	GND
3 LED	Digital pin 2
4 S-GND	GND
5 Vo	Analog pin 0
6 Vcc	5V

<그림 7: ATTiny85와 GP2Y1010AU 핀 구성>

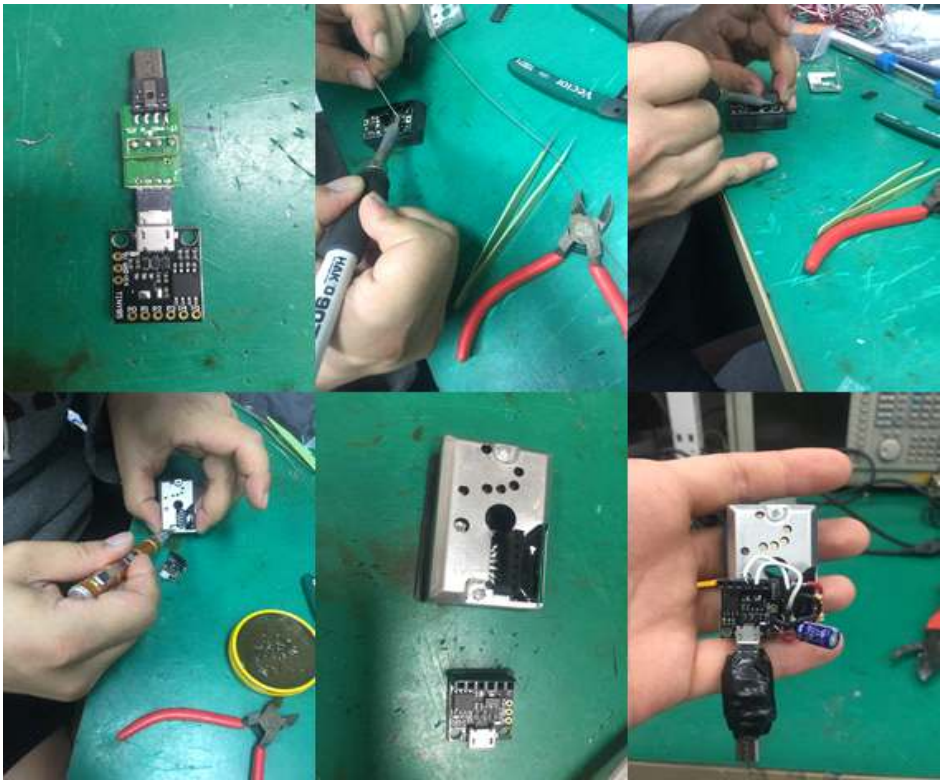
o 먼지센서 데이터 수집을 위한 아두이노 스케치는 다음과 같다.

```
#include "DigiKeyboard.h"
int ledPower = 1;
int measurePin = A1;
int samplingTime = 280;
int deltaTime = 40;
int sleepTime = 9680;
int voMeasured = 0;
float calcVoltage = 0.0;
void setup(){
  pinMode(ledPower,OUTPUT);
}
void loop(){
  digitalWrite(ledPower,LOW);
  delayMicroseconds(samplingTime);
  voMeasured = analogRead(measurePin);
  delayMicroseconds(deltaTime);
  digitalWrite(ledPower,HIGH);
  delayMicroseconds(sleepTime);
  calcVoltage = voMeasured * (5.0 / 1024.0);
  DigiKeyboard.println(voMeasured);
  delay(3000);
}
```

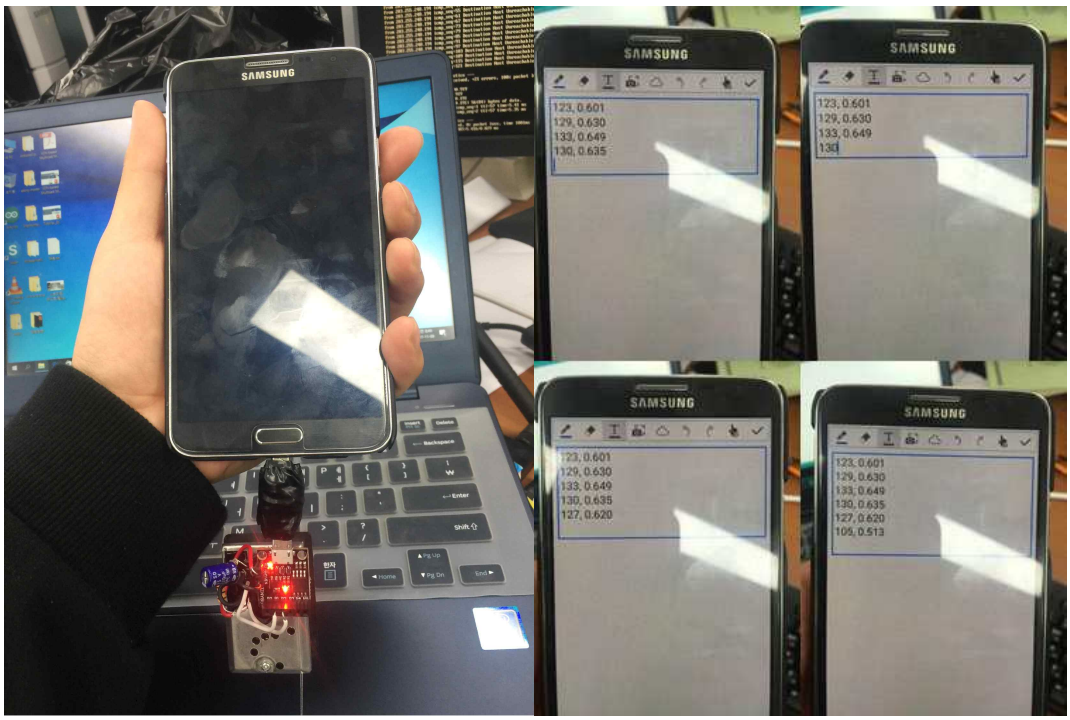
<그림 8: 먼지센서 데이터 수집을 위한 아두이노 스케치>

o 스마트 폰과 ATTiny85 연결을 위한 커넥터 제작한다.

<그림 9: OTG 커넥터>



<그림 10: 스마트 폰 전용 먼지센서 제작 과정>



<그림 11: 완성된 스마트 폰 전용 먼지센서 및 센서 값 확인>

o ATTiny85와 먼지센서를 연결한다.

o 완성된 센서와 스마트 폰을 연결한다.

3.3. GPS 센서 데이터 수집

o 스마트 폰의 위치를 파악하기 위해 내장 된 GPS 센서 값을 가져온다. 안드로이드용으로 작성하였고 전체 소스코드는 Github 저장소에 있다. 사용 예제는 다음과 같다.

```
KGPSInfo gps;
gps = new KGPSInfo(MainActivity.this);
if(gps.isGetLocation()){
   double lat = gps.getLatitude();
   double lng = gps.getLongitude();
}else{
   gps.showSettingsAlert();
}
```

<그림 12: 스마트 폰 내장 GPS 위도, 경도 가져오는 예제>

3.4. 수집한 데이터 HTTP POST

o 수집한 먼지 센서의 값과 GPS 센서의 값을 JSON 형식으로 파싱한다.

o 파싱한 데이터를 metric에 넣고 정해진 ip의 서버로 HTTP POST로 전송한다.

```
public class JSONTask extends AsyncTask<String, String, String> {

   @Override
   protected String doInBackground(String... urls) {
    try {
         JSONObject jsonObject = new JSONObject();
         jsonObject.accumulate("id", metric.getId());
         jsonObject.accumulate("latitude", metric.getLatitude());
         jsonObject.accumulate("longitude", metric.getLongitude());
```

```
        jsonObject.accumulate("dust", metric.getDust());

        HttpURLConnection con = null;
        BufferedReader reader = null;

        try {
                URL url = new URL(urls[0]);
                con = (HttpURLConnection) url.openConnection();
                con.setRequestMethod("POST");
                con.setRequestProperty("Cache-Control", "no-cache");
                con.setRequestProperty("Content-Type", "application/json");
                con.setRequestProperty("Accept", "text/html");
                con.setDoOutput(true);
                con.setDoInput(true);
                con.connect();

                OutputStream outStream = con.getOutputStream();
                BufferedWriter writer = new BufferedWriter(new OutputStream
Writer(outStream));
                writer.write(jsonObject.toString());
                writer.flush();
                writer.close();

                InputStream stream = con.getInputStream();

                reader = new BufferedReader(new InputStreamReader(stream));

                StringBuffer buffer = new StringBuffer();

                String line = "";
                while ((line = reader.readLine()) != null) {
                     buffer.append(line);
                }
                return buffer.toString();
        }
```

```
            } catch (MalformedURLException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                if (con != null) {
                    con.disconnect();
                }
                try {
                    if (reader != null) {
                        reader.close();
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            } catch (Exception e) {
                e.printStackTrace();
            }

            return null;
        }

        @Override
        protected void onPostExecute(String result) {
            super.onPostExecute(result);

        }
    }
```

<그림 13: 센서 데이터 파싱 후 HTTP POST 예제>

3.5. High Level Producer를 통한 데이터 전송

```
var express = require('express');
var app = express();
var kafka = require('kafka-node');
var HighLevelProducer = kafka.HighLevelProducer,
            client = new kafka.Client('192.168.200.4:2181'),
            producer = new HighLevelProducer(client);
```

```
producer.on('ready',function(){
  console.log("Producer on");
});

producer.on('error',function(err){console.log(err)});

app.post('/metric',(req,res) => {
  var inputData;

  req.on('data',(data) => {
      inputData = JSON.parse(data);

      var id = inputData['id'];
      var lat = inputData['latitude'];
      var lon = inputData['longitude'];
      var dust = inputData['dust'];
      var msg = '{\"ID\":\"'+id+ '\", \"latitude\":' + lat + ',
  \"longitude\":' + lon + ',\"dust\":' + dust + '}';

      payloads= [{topic:'KU01_phone',messages:msg}];
      producer.send(payloads,function(err,data){
            console.log(payloads);
      });
  });
  req.on('end', () => {
  });
  res.write("OK!");
  res.end();
});
app.listen(8000,() => {
  console.log('listening on port 8000');
});
```

<그림 14: node.js 서버 소스>

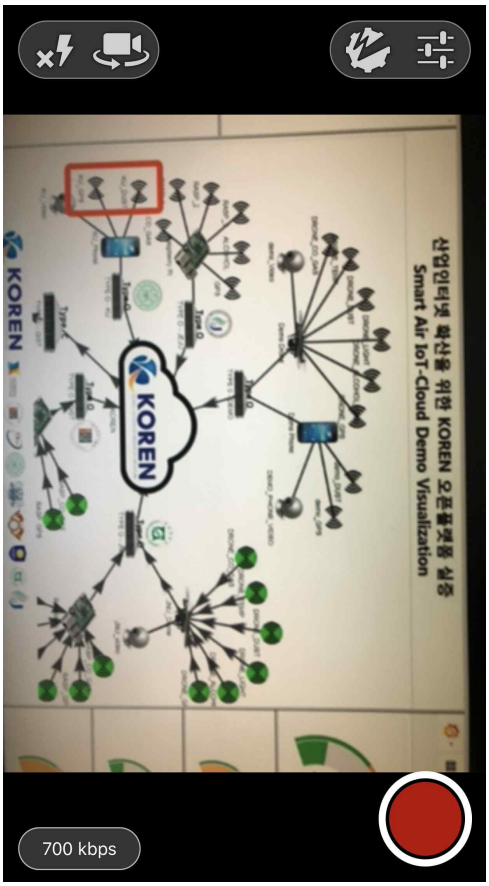
o 3.4에서 HTTP POST로 전송한 데이터를 node.js로 구현한 서버에서 받아 다시 데이터 포맷을 맞춰 Kafka High Level Producer를 이용하여 Kafka 브로커로 전송

4. Smart Air IoT-Cloud 응용 서비스를 위한 비디오 스트리밍

4.1. 개요

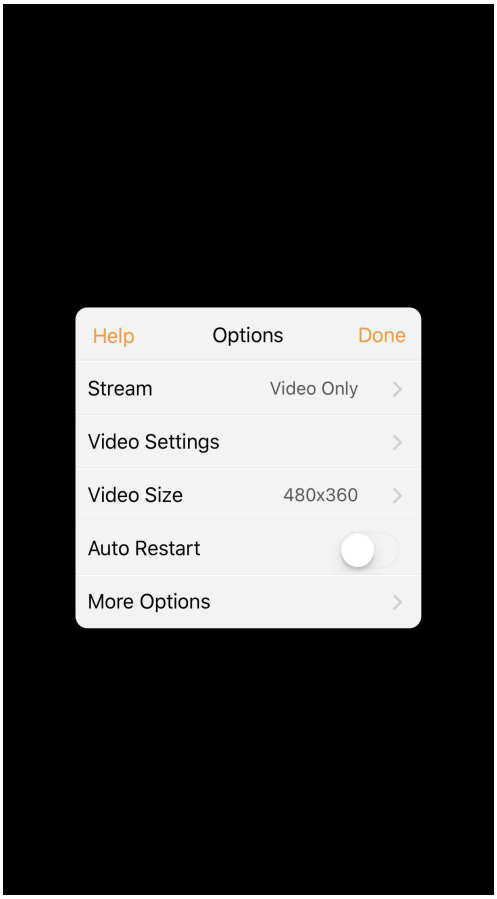
o Smart Air IoT-Cloud 서비스 실증 데모를 위해 이동형 IoT기기(스마트 폰) 전용 애플리케이션을 개발한다. 안드로이드용 애플리케이션으로 작성하였고, 실시간 비디오 스트리밍 전송 기능을 한다. 실시간 비디오 전송은 촬영 중인 영상을 KOREN SmartX Platform 상의 Control Room으로 전송하여 볼 수 있도록 한다.

4.2. 사용 방법



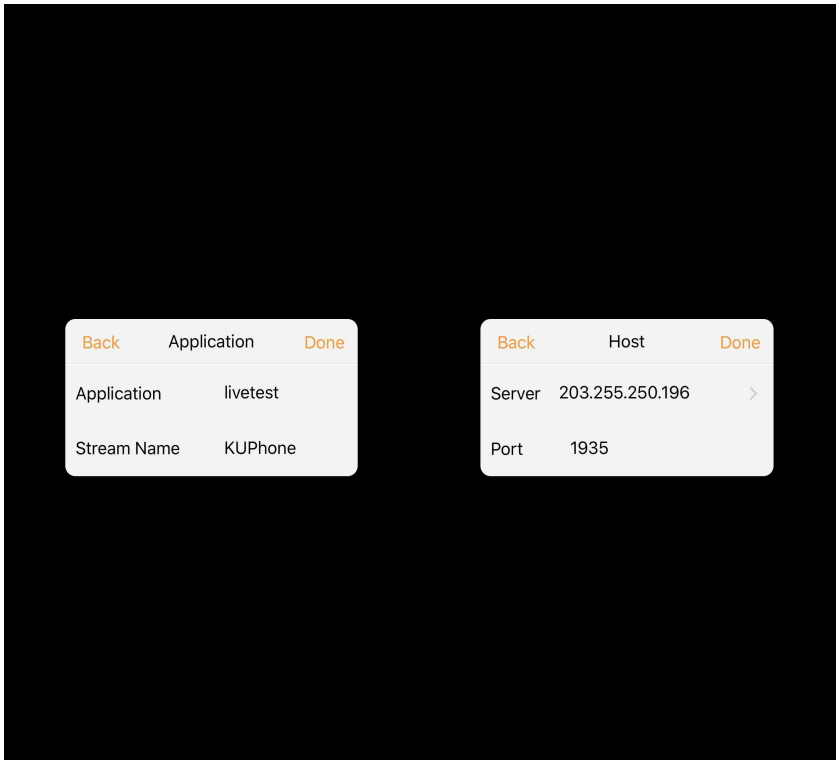
<그림 15: 어플리케이션 메인 화면>

o 어플리케이션을 실행하면 상단에 위치한 4개의 아이콘과 우측 하단에 보이는 녹화 버튼이 존재



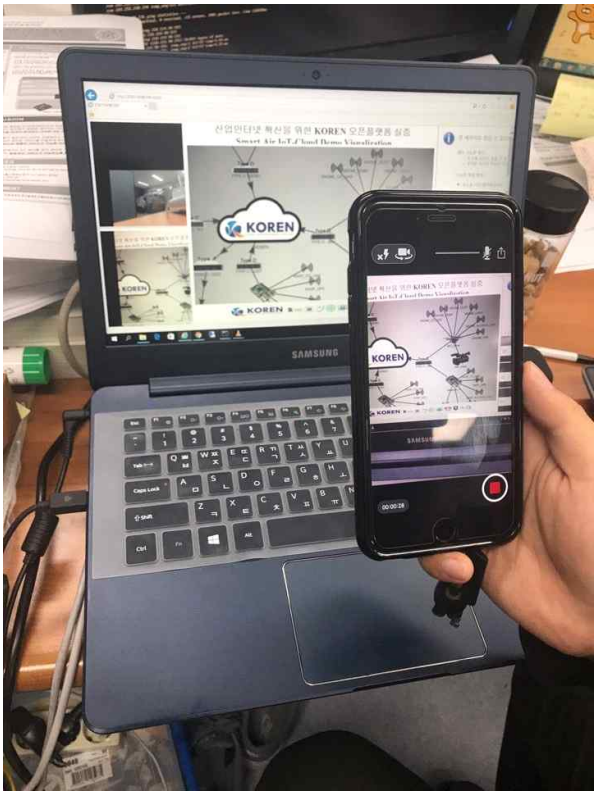
<그림 16: 설정 아이콘 화면>

o 우측 상단에 위치한 설정 아이콘 버튼을 누르면 나오는 화면
- Stream : 영상 + 음성, 영상, 음성 선택 가능
- Video Settings : Framerate, Keyframe Interval 등 설정
- Video Size : 영상 크기 조절



<그림 17: 설정 아이콘 화면2>

o 스트리밍 url: rtmp://$Server:$Port/$Application/$StreamName
  - ex) rtmp://203.255.250.196:1935/livetest/KUPhone



<그림 18: 스트리밍 테스트>

o 스마트 폰 어플리케이션으로 촬영 중인 스트리밍 영상을 Visualization Center에서 확인